

## APPENDICES

## APPENDIX A

The table below shows the packages and classes in JXDB system, named **xdb**.

Table A-1: JXDB Packages (xdb)

XDB sub-packages	Description
xdb.io	This package consists of TextFile class, which is designed to read any input text file, such as the required configuration files.
xdb.operator	<p>This package consists of XQuery operators and functions in JXDB, which are needed to generate XQuery expressions using XQuery wizard. Currently, these operators and functions are sub-sets of what the third-party XQuery supports in order to demonstrate the ability of JXDB to incorporate XQuery engine to perform XQuery facilities.</p> <p>The supported operators and functions classes are: <i>and</i>, <i>or</i>, <i>contains</i>, <i>average</i>, <i>count</i>, <i>min</i>, <i>max</i>, <i>date</i>, <i>float</i>, <i>empty</i>, <i>distinct values</i>, <i>=</i>, <i>&gt;</i>, <i>&gt;=</i>, <i>&lt;</i>, <i>&lt;=</i>, <i>!=</i>, <i>position</i>, <i>sum</i>, <i>not</i>, <i>starts with</i> and <i>ends with</i>.</p>
xdb.sql	This package consists of the required database table builder adapters. These database table builder adapters are designed for JXDB to connect to different database vendors and perform mapping of classes between Java data types and database data types during the transfer between different databases due to the incompatibility of different data types supported by these databases.
xdb.ui	This package consists of all the UI classes for generating and rendering JXDB graphical user interfaces.

## APPENDIX B

This appendix describes README.txt for JXDB system (version 1.1).

### Introduction

This is a prototype of JXDB system version 1.1. This release is one of the pioneers of GUI-based XML middleware in the market that is able to perform the integration between XML and Relational Databases using XML XQuery. JXDB is distributed in a JAR format. This JAR file consists of all the required classes and essential libraries to run JXDB system, such as Borland JBuilder libraries, and JDBC drivers for Oracle 9i, Microsoft SQL 2000 and MySQL in JAR files. Additionally, it is also inclusive of a third-party XML XQuery engine, thus JXDB is optimised to generate and process XML XQuery queries.

### Quick Start

Create a directory named xdb in C drive (or your preferred drive) and unzip the xdb.zip file into this directory, for example: C:\xdb. Then execute "Run.cmd" (for Windows platform) from this installation directory (C:\xdb) to launch the graphical user interface of JXDB. If this does not work, please check the following:

1. Do you have a Java runtime environment installed? Minimum JRE requirement is version 1.3 and above.
2. Is java.exe located in your classpath? Please refer to Sun website on how to setup Java and other required libraries classpath in your environment.

3. Does your version of Windows use “.cmd” files or “.bat”? Should the latter be the case, then rename the “.cmd” extension to “.bat”. Error messages can be analyzed more easily if you were to run JXDB from the command line.

## **Prerequisites**

You must have a Java runtime environment installed on your machine. JXDB was successfully tested with JDK 1.4.1. In addition, to fully run JXDB, you must have the following relational databases installed before you can run JXDB. JXDB supports the following databases:

1. Oracle 9i
2. Microsoft SQL 2000
3. MySQL version 4.0.15 and above. (JXDB has been tested with this version)
4. Microsoft Access 2002

For installation of these databases, please refer to their official websites or user installation guide. If you only have installed Microsoft SQL 2000, then you cannot connect to Oracle 9i or MySQL databases. Unless you install all the databases, only then you can fully manipulate data integration between different databases. Also, ensure that the following libraries (JAR files) are installed before you start JXDB (p/s: they are already included in the jxdb.jar):

1. Java Runtime Environment 1.3 and above. (Fully tested with JDK 1.4)
2. JDBC Driver for Oracle 9i
3. JDBC Driver for Microsoft SQL 2000
4. JDBC Driver for MySQL
5. ODBC-JDBC Driver (It is bundled together in Sun JDK 1.4 and above).
6. XQuery engine library, i.e. Quip.jar



## User Installation (First Time Installation)

You have already unpacked the distribution “.zip” file into your desired directory (For example -> C:\xdb). Next, execute “Run.cmd” from that folder to start JXDB system. It is as simple as that! In summary, below are simple installation steps for first time installation:

1. Create a directory of your own choice named xdb on any preferred drive.
2. Unzip “xdb.zip” file into the created installation directory or home directory.
3. The installation directory structure should look like this:
  - a) work (C:\xdb\work) – to store all the generated XML data files.
  - b) config (C:\xdb\config) – to store all the JXDB configuration files.
  - c) map (C:\xdb\config\map) – to store Java data types mapping files.
  - d) lib (C:\xdb\lib) – to store Quip libraries and Quip XQuery engine, e.g. quip.exe.
  - e) Run.cmd, jxdb.jar, jxdb.exe, and README.txt.

Note: By default, the home directory is in C:\xdb. If you were to change this directory path value, for example change to E:\xdb, please remember to update the variable “JXDB\_HOME” in Run.cmd file to this new value.

## First Step to Run

Open a command shell (also known as a command prompt). Change to the directory where you found this “ReadMe.txt” file after you have unzipped the files (example, go to c:\xdb directory). Execute the command “./Run.cmd”. You will see a GUI-based Java program started, which is a GUI front-end for JXDB. At this moment, you can start using the JXDB system. Please ensure that:

- All the installed databases are up and running, before you try to connect to any of these databases via JXDB.

- When transferring XML data to any specific database use a dedicated customised database driver and its database table builder adapter to perform the transfer process. This is essential in order to perform data types or schema mapping between different database vendors because of the different data types supported by these databases. For example, to transfer XML data to Oracle 9i database, select JDBC Driver for Oracle and its respective Oracle database table builder adapter, whereas to transfer to Microsoft SQL 2000 database, select JDBC Driver for SQL 2000 and its respective SQL 2000 database table builder adapter. The same concept also applies when transferring to MySQL database. Even though JXDB provides ODBC-JDBC driver to connect and access data from Oracle 9i, Microsoft SQL 2000 and MySQL databases, but ODBC-JDBC driver cannot be used to transfer XML data to any of these databases unless you have customised a dedicated database table builder adapter for this driver. This is mainly because of the different data types supported by these different database vendors and the table builder adapters that are used to perform mapping of data type schemas between these databases during XML integration process.

### **Limitations of XQuery Features**

There are a number of XQuery features not yet implemented in the current version. This is because of the current immature status of XQuery and likely will remain the same until a number of unresolved issues have been resolved by W3C XQuery working group or wait till XQuery get W3C recommendation status in the near future. Most importantly, the implemented features of XQuery in JXDB are greatly influenced by the compatibility and

supported features of a third-party component. i.e. Quip engine. The following are the limitations of the features:

- Supports FLWOR expressions and result sets only return element and attribute nodes and integers.
- Supports most of the W3C XQuery use cases, especially the "Use Case R - Access to Relational Data".
- The return element nodes cannot have the word "text" as the element name as this will cause the XQuery engine to throw exception because the keyword "text" is used by the engine.
- Users cannot declare their own functions or sort or order by at the moment, the actual implementation still not standardised yet.
- Operators such as "union" and "intersection" are not yet implemented. (Although "and" and "or" are implemented due to XQuery engine has not supported yet).
- No support for generated QNames for constructed nodes.
- No support for dereference symbol.
- No support for static type checking.
- Only XPath's abbreviated syntax is supported, but users cannot do parent ("..").
- Other features are only partially implemented due to Quip compatibility and limitations.
- The list of supported built-in functions are: contains(), starts with(), ends with(), count(), avg(), sum(), empty(), position(), distinct values(), max(), min(), float(), date(), and, or, not, =, !=, <, <=, >, >=.

# APPENDIX C

Listed below are simple instructions on how to run JXDB system:

1. Execute the “Run.cmd” file to launch JXDB system.
2. Click on “XML Files” tree to expand and display its leaf nodes. These XML data files are pre-generated XML samples for testing purposes. JXDB will display these data in data grid control format on Data Grid tab and also in tree format on XML tab. Please refer to the following Figure C-1 and Figure C-2.

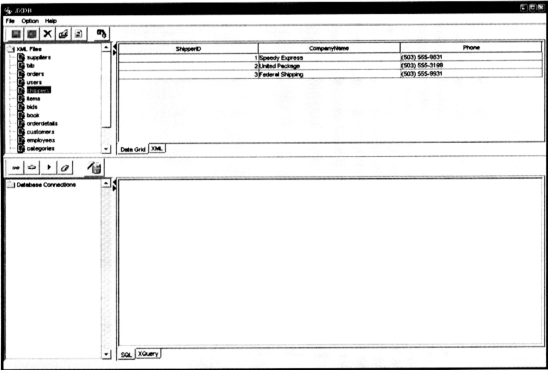


Figure C-1: Displaying Data on Data Grid Control

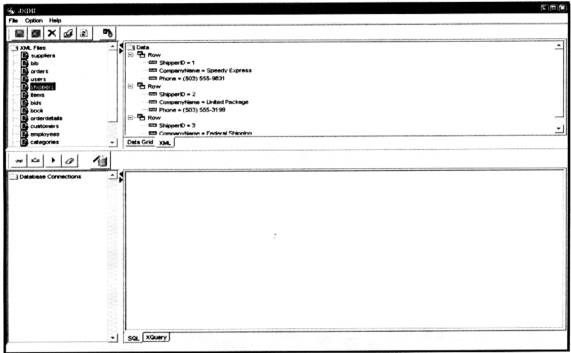



Figure C-2: Displaying Data in Tree Format

- Click on the  button to establish a database connection. You will see a pop-up “New Database Connection” dialog box. Firstly, try to connect to Oracle 9i database, select Oracle JDBC driver from the driver drop-down list. Secondly, select its respective connection string from the connection string drop-down list. You can easily add new JDBC driver details by modifying the Driver.ini file. Besides that, you can construct a new or modify existing connection string based on your choice of database, port or database URL in the Connection.ini file (these files are located in the config folder). For example, to connect to Oracle 9i database, the JDBC driver name used is **oracle.jdbc.OracleDriver** and its respective connection string constructed is **jdbc:oracle:thin:@localhost:1521:JXDB**; where the database server name is **localhost**, the default port used is **1521** and the database name to be connected is **JXDB**. Whereas in the case of Microsoft SQL 2000, the JDBC driver name used is **com.microsoft.jdbc.sqlserver.SQLServerDriver** and its respective connection string

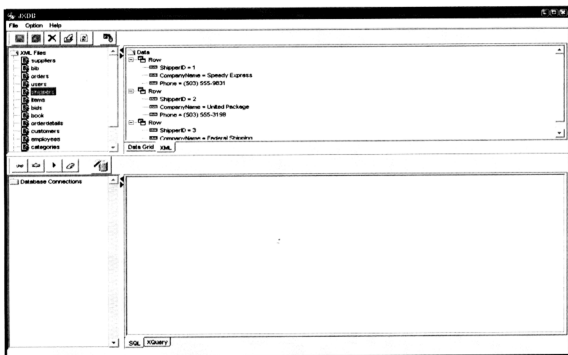



Figure C-2: Displaying Data in Tree Format

- Click on the  button to establish a database connection. You will see a pop-up “New Database Connection” dialog box. Firstly, try to connect to Oracle 9i database, select Oracle JDBC driver from the driver drop-down list. Secondly, select its respective connection string from the connection string drop-down list. You can easily add new JDBC driver details by modifying the Driver.ini file. Besides that, you can construct a new or modify existing connection string based on your choice of database, port or database URL in the Connection.ini file (these files are located in the config folder). For example, to connect to Oracle 9i database, the JDBC driver name used is **oracle.jdbc.OracleDriver** and its respective connection string constructed is **jdbc:oracle:thin:@localhost:1521:JXDB**; where the database server name is **localhost**, the default port used is **1521** and the database name to be connected is **JXDB**. Whereas in the case of Microsoft SQL 2000, the JDBC driver name used is **com.microsoft.jdbc.sqlserver.SQLServerDriver** and its respective connection string

or also known as database URL is **jdbc:microsoft:sqlserver://localhost;DatabaseName=JXDB**; where the parameter of the database server name is **localhost**, and the database name to be connected is **JXDB**. The JDBC driver name used for MySQL is **com.mysql.jdbc.Driver** and its connection string is **jdbc:mysql://localhost:3306/JXDB**; where the database server name is **localhost**, the default port used is **3306** and the database name to be connected is **JXDB**.

4. Enter an authorised user name and a password, to access the selected database, in this case for example, assuming user “demo” has been created and assigned necessary security credentials in Oracle 9i before performing this step. Also, prior to connecting to Oracle 9i database, we assume that you have created a database instance named “JXDB” in Oracle 9i database or any other database server that you would like to connect to. Click “OK” button to start connecting to database. Please refer to the figure shown below:

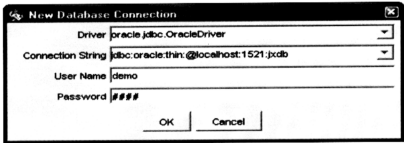







Figure C-3: Connecting to Oracle 9i Database Dialog


5. After successfully connected to the selected database, “Database Connections” tree will add this connection to its tree node. Select the connected connection node, then enter a SQL statement to retrieve data from the connected database. However, prior to this, you must remember that you can only retrieve data from the “JXDB” database instance



(created in step 4) if the respective database has some data inside. Additionally, you can upload some sample data to the JXDB database, such as Bids.xml, Items.xml and Users.xml (located in one of the sub-folders in the installation folder, named “work”. These sample data are taken from W3C XQuery Use Case samples). To upload these sample data, proceed to step 13 before going to step 6.


6. Enter a SQL Select statement at the SQL Editor TextArea, for example: “Select \* from Users”, and click on the  button to execute the SQL statement. Continue to retrieve data from Bids and Items table. These returned data are stored as XML files and embedded inside are its built-in mapping attributes. These generated XML files are stored in the work folder.
7. To clear SQL Editor TextArea or XQuery Editor TextArea, click on the  button.
8. To remove or disconnect an existing database connection, select the database connection you want to remove and then click on the  button to disconnect the database connection.
9. To remove any unwanted generated XML file, select the unwanted XML file and then click on the  button to delete the physical XML file from local FileSystem.
10. To clear or delete all unwanted XML files, instead of deleting the file one by one as described in the previous step 9, you can delete or clear all these XML files all at once by clicking on the  button. This is dangerous as it will delete permanently all your



XML files in the work folder, but you can re-generate these files by performing step 6 provided you still have the active database connection.

11. To refresh the list of files in the work folder after deleting or clearing those unwanted XML files from step 9 or 10, click on the  button to refresh these files.

12. You can modify any XML data via the data grid control by selecting the required XML file from the XML tree, then click on the value of the cell you want to modify and enter the new value. Before proceeding with another transaction, you can save the modified data by clicking on the  button. Similarly, you can also click on the  button which will save all the unsaved data you made previously across several XML files just by one single click. Once these modified data are saved, you can start transferring these modified XML data across to different databases.

13. To run XQuery Wizard, click on the  button to launch the XQuery Builder Wizard dialog box. Please refer to Figure C-4. You can construct your choice of XQuery expressions (FLWOR) by selecting the XML files, its respective columns, operators and functions from their respective drop-down lists. Once you have finished building these XQuery expressions, click on “OK” button to close the wizard. For further details on XQuery sample queries, please refer to APPENDIX D.

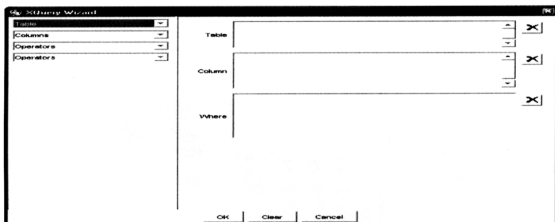



Figure C-4: XQuery Wizard

14. Finally, you can perform data transfer process to upload these XML data across heterogeneous relational databases. Prior to this, remember to first select the correct connection node of the database that you would want to transfer to, then click on the  button. Upon clicking on this button, you will see a pop-up Table Builder Adapter dialog box. Then, click on the drop-down list to select the correct table builder adapter for the respective database. For example, to transfer to Oracle 9i database, you need to select JDBC driver for Oracle and its respective table builder adapter is **xdb.sql.Oracle90TableBuilder**. Otherwise, in the case of Microsoft SQL 2000, you need to select JDBC driver for Microsoft SQL 2000 and its respective table builder adapter is **xdb.sql.SQL2000TableBuilder**. Whereas, to transfer to MySQL database, select JDBC driver for MySQL and its respective table builder adapter is **xdb.sql.MySQLTableBuilder**. If you were to add a new JDBC driver for a specific database, such as Sybase, you will need to create and customise a dedicated table builder adapter for this driver before you can perform data transfer to upload these XML data to the newly added database.

## APPENDIX D

*Besides building your own XQuery expressions, you can copy and paste these sample queries to the XQuery Editor TextArea, then click on the ▶ button to execute the XQuery expression.*

Here, these XQuery syntaxes conform to the features and constraints

implemented by the Quip XQuery engine, and it might be differ slightly from W3C XML

XQuery standards as currently XML XQuery still holds working draft status. Hence, each

vendor has its own XQuery implementation until W3C resolves the issues and starts to

standardise XQuery language. Several XQuery sample queries are listed below.

1. List books published by MS Book Ptd after 1999-12-31, including all their fields.

**For** \$t1 in document("book.xml")//Row

**Let** \$r1 := \$t1/Publisher, \$r2 := \$t1/Date, \$r3 := \$t1/BookName

**Where** \$r1 = "MS Book Ptd" and \$r2 > "1999-12-31"

**Return**

<Row> { \$r1 }

{ \$r2 }

{ \$r3 }

</Row>

2. List the total count of suppliers in suppliers.xml.

```
For $t1 in document("suppliers.xml")
```

```
Return
```

```
<Row>          <count> { $r1 } </count> </Row>
```

3. For each book in the book.xml, list the name of the book and author, grouped inside a result element.

```
For $t1 in document("book.xml")//Row
```

```
Return
```

```
<Row>
```

```
  { $t1/BookName }{ for $a in $t1/Author return $a }
```

```
</Row>
```

4. For each bid in the bids.xml, list the UserID, ItemNo, Bid, Bid Date and Sum of Bid Item.

```
For $t1 in document("bids.xml")//Row
```

```
Let $r1 := $t1/UserID, $r2 := $t1/ItemNo, $r3 := $t1/Bid, $r4 := $t1/Bid_Date, $r5 :=  
float($r2/text()) + float($r3/text())
```

```
Return
```

```
<Row>          { $r1 }
```

```
                { $r2 }
```

```
                { $r3 }
```

```
                { $r4 }
```

```
                <SumOfItemBid>{ $r5 }</SumOfItemBid> </Row>
```

5. For each book found in book.xml and bib.xml, list the name of the book, author and its price from each source; extracting from multiple XML sources.

```
For $t1 in document("book.xml")//Row, $t2 in document("bib.xml")//Row
Where $t1/BookName = $t2/BookName
Return
<Row> <BookName>{$t1/BookName/text()}</BookName>
      <Author>{$t2/Author/text()}</Author>
      <Price-bn>{float($t1/Price/node())}</Price-bn> </Row>
```

6. For each item found in bids.xml and items.xml, list the UserID and its Reserve Price where UserID contains "U01" and ItemNo from items.xml, which is the key identifier equals to ItemNo from bids.xml; extracting from multiple XML sources based on their relationships or keys.

```
For $t1 in document("bids.xml")//Row, $t2 in document("items.xml")//Row
Let $r1 := $t1/UserID, $r2 := $t2/ItemNo, $r3 := $t2/Reserve_Price, $r4 := $t1/ItemNo
Where contains($t1/UserID/text(), "U01") and $r2 = $r4
Return
<Row> { $r1 }
      { $r3 }
</Row>
```